Hybrid interconnection topologies for high performance and low hardware cost based on hypercube and k-ary n-tree

Hybrid interconnection topologies for high performance and low hardware cost based on hypercube and k-ary n-tree

Junhong Li*
Graduate School of CIS
Hosei University
junhong.li.2k@stu.hosei.ac.jp

***Abstract*—The implementation of fat-tree interconnection networks is prevalent in high-performance parallel computing. However, the traditional fat-tree structure requires a considerable amount of switches and links to connect computing nodes, resulting in a significant increase in hardware costs for large-scale high-performance systems. This study proposes two innovative hybrid topologies, the k-ary n-tree k-cube (KANTC) and the Mirrored k-ary n-tree k-cube (MiKANTC), to address the aforementioned issue. The proposed topologies merge the characteristics of the hypercube and fat-tree structures. Instead of the traditional direct connection of k computing nodes to an edge-level switch, the edge-level switches in the fat-tree are substituted with k-cubes. This results in the formation of k^{n-2} k-cubes at the edge level, where each k-cube links k switches to the upper level of the k-ary n-tree, while the remaining switches link to the compute nodes. Hence, all the cubes are capable of interconnecting $k(2^k-k)$ compute nodes. Shortest path-based routing algorithms are proposed for these hybrid topologies, and several link fault tolerant routing algorithms are developed to enhance the fault tolerance of the entire topology. The proposed hybrid topologies are then evaluated in terms of path diversity, cost, and performance. The results demonstrates that the proposed KANTC and MiKANTC topologies exhibit improved performance, with up to 84% reduction in the number of switches and 78% reduction in links in large parallel systems when $k = n = 8$, compared to the conventional fat-tree topology. Additionally, these hybrid topologies display enhanced path diversity compared to traditional fat-tree.**

1. Introduction

The deployment of high-performance computing systems requires the integration of large-scale interconnection networks. As the number of compute nodes in these systems increases with the advancements in distributed and cloud computing technologies [1], it becomes imperative to strike a balance between cost and performance. This has led to the design of various interconnection networks aimed at optimizing this trade-off. One of the most widely used interconnect topologies in high-performance computing systems is the fat-tree [2], which is used in top500 supercomputers such as the Summit [3]. The fat-tree network separates traffic between compute partitions and storage subsystems, thus offering a more predictable application performance. Furthermore, its high level of redundancy and reconfigurability guarantee reliable performance even in the event of network component failures [4]. In contrast to traditional tree network topologies, where bandwidth converges level by level, with the root having a smaller bandwidth compared to the sum of all leaf bandwidths, the fat-tree resembles a real tree with thicker branches closer to the root. This prevents bandwidth convergence and enables the support of non-blocking networks. However, the scalability of the traditional fat-tree is limited by the number of ports in the core layer switches, which poses a hindrance to the long-term development of data centers.

To address this limitation, various fat-tree schemes with multiple roots have been proposed, such as the k-ary n-tree , represented by the acronym KANT, proposed by Petrini and Vanneschi [5]. The value of k in this scheme represents both the number of links to upper or lower layers and the number of compute nodes connected to a single leaf switch, while n represents the number of layers. This structure allows for a fixed number of switch ports regardless of the size, making it highly scalable. In their studies [6], [7], Gómez et al. introduced the concept of Reduced Unidirectional Fat-Tree (RUFT) structure, which minimizes the number of switches required in the system by implementing a unidirectional flow of packets from the first-level switch to the last-level switch and onward to the compute node. The utilization of unidirectional links in RUFT has contributed to the reduction of hardware costs. Ludovici et al. [8] demonstrated that for implementing network-on-chip (NoC), RUFT is a more powerful solution than conventional butterfly. Wang et al. tackled the issue of complicated floor plan design in fat-tree-based NoC and presented an optimized solution. This solution aimed to minimize the amount of intersection points and the distance of links, and thus improve the efficiency of the fat-tree design [9]. Li proposed the MiKANT topologies, a mirrored version of KANT, as a more hardware-efficient alternative to bidirectional Clos network and fat-tree [10]. MiKANT reduces hardware costs by connecting a larger number of compute nodes to a smaller number of switches and links and improves communication time and performance by reducing the average distance. A link fault-tolerant routing algorithm in the MiKANT network was also presented by Wang in [11], and its performance was evaluated through small-scaled simulation.

The use of hypercube [12] structures in networking is widespread due to its desirable topological properties and its ability to emulate various other commonly utilized networks. Despite its popularity, traditional hypercube networks face a major disadvantage: The communication links per node scale proportionally to the logarithm of the number of nodes in the network, which impedes the scalability of the network for large-scale systems. To address this issue, Arai and Li proposed a variant topology based on the hypercube called the Generalized-

* Supervisor: Prof. Yamin Li

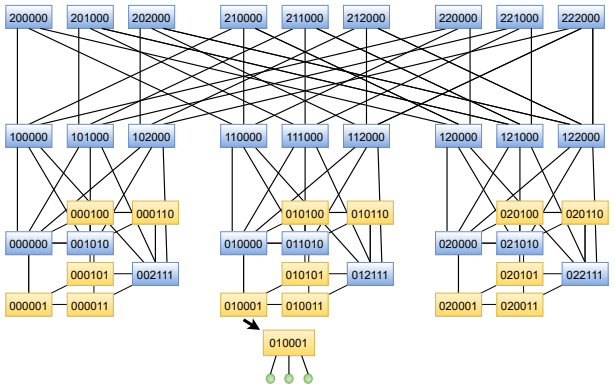

Figure 1: A 3-ary 3-tree 3-cube

Star cube (GSC) [13], and presented a routing algorithm for it. Additionally, Wang proposed a hybrid structure that combines elements of hypercube and fat-tree [14], and presented a method for evaluating path diversity in this architecture.

To address the challenges faced by traditional hypercube networks in large-scale scaling, we present two novel network topologies, namely the k-ary n-tree k-cube (KANTC) and the Mirrored k-ary n-tree k-cube (MiKANTC), that are based on a combination of n-dimensional cube and fat-tree. The design of the KANTC/MiKANTC topologies involves the use of hypercubes to replace the edge-level switches of KANT, where the value of k denotes the dimension of the hypercube as well as the switch array. As a result, each switch in the architecture is capable of maintaining $2k$ links. The performance of KANTC and MiKANTC was evaluated in terms of the number of compute nodes, links, switches, and diameters, as well as cost ratio, cost performance, and path diversity. The results indicate that KANTC and MiKANTC exhibit higher path diversity and lower hardware costs compared to traditional fat-tree topology.

2. KANTC and MiKANTC

This section describes how KANTC and MiKANTC are encoded and their topological characteristics.

2.1. KANTC

The proposed topology, denoted as KANTC(k, n), combines features of the hypercube as well as the KANT, hence the name "k-ary n-tree k-cube." The parameter k signifies the hypercube's dimensionality and the array of the KANT. The architecture of KANTC is composed of two distinct sections: the top section, which comprises of $n-1$ layers of a KANT, and the bottom section, which is comprised of k^{n-2} hypercubes. In contrast to conventional KANT, where the switches at the edge level directly connect to the k computing nodes, the edge-level switches in KANTC are replaced by hypercubes. Each hypercube has the capacity to support connections to $k(2^k - k)$ computing nodes. We propose a hybrid encoding approach for the representation of KANTC(k, n), which combines the properties of both the KANT and the hypercube. Each switch in the KANTC(k, n) structure is marked with a triplet $\langle L, T, C\rangle$, as illustrated in Fig. 1.

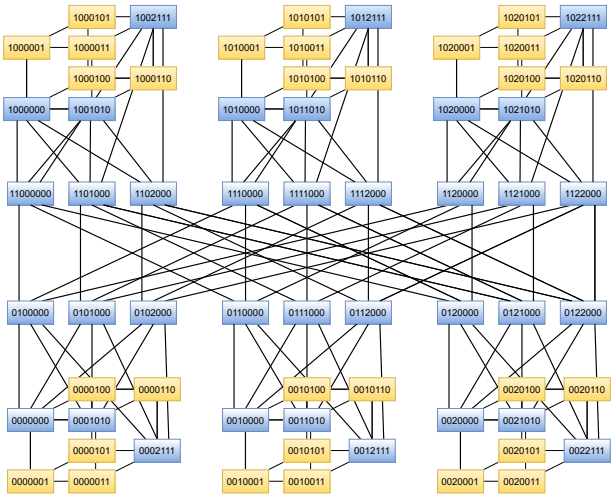

Figure 2: A Mirrored 3-ary 3-tree 3-cube

The binary encoding of the hypercube switch, represented by $C = C_{k-1}, C_{k-2}, ..., C_1, C_0$, is expressed using k bits, where each C_i is either 0 or 1.

To distribute the intermediate switches evenly, we employ a bitwise inverse approach. The encoding of the first $\lceil k/2 \rceil$ switches is determined by picking all zeros as the first switch. The encoding for the subsequent switches is then obtained by inverting the values from bit i to bit $k-i$ of the previous switch. The remaining $\lfloor k/2 \rfloor$ switches are obtained by inverting the previously determined switches. For instance, in the case of a four-dimensional hypercube, the intermediate switches are represented by $\langle 0, 0, 0, 0\rangle$, $\langle 0, 1, 1, 0\rangle$, $\langle 1, 0, 0, 1\rangle$, $\langle 1, 0, 0, 1\rangle$ $\langle 1, 1, 1, 1\rangle$. The part $\langle L, T\rangle$ of the KANTC encoding is consistent with the KANT, while the part $\langle C\rangle$ is determined by the hypercube. In the higher layers of the KANT ($n-1$ to 1), $\langle C\rangle$ remains all-zero, while the encoding in the lowest layer (0th layer) is aligned with the hypercube.

In each 3-cube of the KANTC$(3, 3)$ structure, there are $2^k = 8$ switches, including the intermediate switch. The intermediate switches connect the 3-cube to the KANT, while the other switches connect to the compute nodes, with each switch connecting to three computational nodes. Fig. 1 shows an example of a KANTC$(3, 3)$ topology, where the lower part consists of 3 3-cubes and the upper part consists of the KANT ,there are 45 compute nodes in total.

2.2. MiKANTC

A MiKANTC(k, n), a hybrid topology that merges the properties of both a MiKANT and a hypercube, has been devised. Each switch in the MiKANTC(k, n) is tagged with a unique identifier made up of four elements: group (G), level (L), switch ID within the k-ary n-tree of group G (T), and ID within the k-cube (C).

For instance, MiKANTC $(3, 4)$ comprises of 18 3-cubes and 270 compute nodes. The level 2 switches of group 0 and group 1 form a KANTC $(3, 4)$, and the level 2 switches of group 1 and group 0 form another KANTC $(3, 4)$.

The connections between switches in the MiKANTC are determined based on the switch's level and group. If

TABLE 1: ANALYSIS OF NETWORK TOPOLOGICAL CHARACTERISTICS

Parameters	HC(k)	k-ary n-tree	MiKANT(k,n)	KANTC(k,n)	MiKANTC(k,n)
Nodes	2^k	k^n	$2k^n$	$(2^k-k)k^{n-1}$	$2(2^k-k)k^{n-1}$
Switches	2^k	nk^{n-1}	$(2n-2)k^{n-1}$	$(n-1)k^{n-1}+2^k k^{n-2}$	$(2n-4)k^{n-1}+2^{k+1}k^{n-2}$
Links	$k2^{k-1}$	nk^n	$(2n-1)k^n$	$(n-1)k^n$ $+(2^{k-1}+2^k-k)k^{n-1}$	$(2n-3)k^n$ $+(3\times 2^k-2k)k^{n-1}$
Radix/Degree	k	$2k$	$2k$	$2k$	$2k$
Diameter	k	$2n$	$2n$	$2n+k$	$2n+k$

the level is less than n-2, the switch P represented by

$$\langle G, L, T_{n-2}, ..., T_{L+1}, T_L, T_{L-1}, ..., T_0, C_{k-1}, ..., C_0\rangle$$

links to switches in the next level with the same group, but different switch encoding in T_{L+1}. If the level is equal to $n-2$, the switch connects to switches in a different group

$$\langle \overline{G}, L, *', T_{n-3}, ..., T_1, T_0, C_{k-1}, ..., C_0\rangle$$

where $*'$ takes any value within the interval of [0, $k-1$]. In MiKANTC $(3,4)$, the switch P: $\langle 0,2,1,1,0,0,0,0\rangle$ is connected to switches $\langle 1,2,0,1,0,0,0,0\rangle$, $\langle 1,2,1,1,0,0,0,0\rangle$, and $\langle 1,2,2,1,0,0,0,0\rangle$, which belong to distinct groups. If the present and destination nodes belong to the identical group, MiKANTC operates in the same manner as KANTC.

The topological characteristics of various topologies, including the k-cube, KANT, MiKANT, KANTC, and MiKANTC, are summarized in Table 1. The results demonstrate that, in comparison to conventional KANT and MiKANT, KANTC and MiKANTC demonstrate enhanced computational node connectivity, utilizing fewer switches and links, with only a marginal increase in the diameter of k. In the subsequent section, the performance of the topology will be evaluated in detail.

3. Routing Algorithm

This section presents the shortest path routing algorithm for the KANTC(k,n) and MiKANTC(k,n) topology, which is formulated as a deterministic process.

3.1. Algorithm for Routing in KANTC

In this section, we present the routing algorithm for KANTC (k,n), which is a combination of traditional KANT and hypercube. Firstly, we use a static method to identify all intermediate switches and store their IDs in a list, referred to as I. For the lower half of KANTC, which is the hypercube part, if the source node and the destination node belong to different cube's switches, the routing algorithm determines the shortest path to the intermediate switch. In the event of multiple shortest paths, one is selected at random. If the source and destination nodes are located within the same switch of a cube, the packet will be transmitted directly from the source node to the target node.

In the upper half of the KANTC topology. The algorithm finds the Nearest Common Ancestor(NCA) between the source intermediate switch(SIS) and the destination intermediate switch(DIS). The route from the source switch to the target switch is divided into two parts: from the SIS to the NCA, and from the NCA to the DIS. The process of reaching the NCA may involve multiple paths, but the path from the NCA to the DIS is determinate.

The algorithm for routing in KANTC is determined by the identification of the present switch or node, represented as

$$P = \langle L_P, TP_{n-2}, ..., TP_1, TP_0, CP_{k-1}, ..., CP_0\rangle$$

and the destination node ID, represented as

$$D = \langle L_D, TD_{n-2}, ..., TD_1, TD_0, CD_{k-1}, ..., CD_0\rangle$$

If $TP_{n-2}, ..., TP_{LP} \neq TD_{n-2}, ..., TD_{LD}$, it signifies that the present switch/node has not yet reached the nearest common ancestor, and the network data must be transmitted to a higher layer switch through an intermediate switch or upper interface. Once the present switch/node arrives at the nearest common ancestor, the routing process enters the downward phase and becomes deterministic, with the packet being directed directly towards the destination switch.

During the path to the NCA, a switch P in the L_P level:

$$P = \langle L_P, P_{n-2}, ..., P_{LP+1}, P_{LP}, D_{LP-1}, ..., D_0, C\rangle$$

sends data to switch N, which is closer to D than P:

$$N = \langle L_N, P_{n-2}, ..., P_{LP+1}, D_{LP}, D_{LP-1}, ..., D_0, C\rangle$$

with C representing $C_{k-1}, ..., C_0$, P_{LP} being changed to D_{LP}. In to downward phase, if the L_P of P is equal to 0, the present switch reached the same cube with destination node, and the data is sent directly to the destination node D.

3.2. Algorithm for Routing in MiKANTC

The routing algorithm of MiKANTC is comprised of a combination of MiKANT and hypercube, with the present switch and node encoding represented as

$$P = \langle G_P, L_P, TP_{n-2}, ..., TP_1, TP_0, CP_{k-1}, ..., CP_0\rangle$$

and the encoding of the destination switch and node is represented as

$$D = \langle G_D, L_D, TD_{n-2}, ..., TD_1, TD_0, CD_{k-1}, ..., CD_0\rangle$$

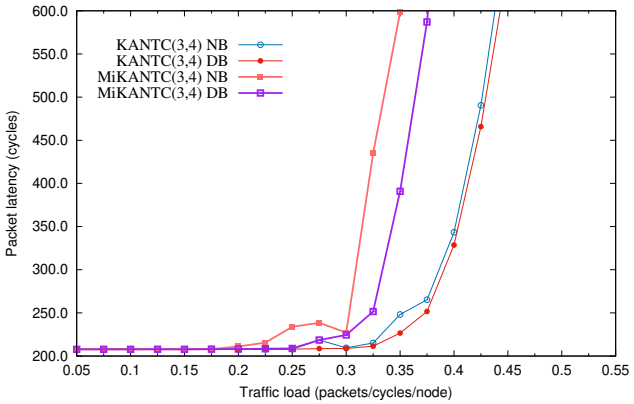

Figure 3: packet latencies of KANTC(3, 4) and MiKANTC(3, 4)

In the case where the present switch and the destination switch are part of the same group, the routing protocol of MiKANTC is equivalent to that of KANTC. Conversely, if the present switch and the destination switch belong to different groups, the network data will first be transferred to the $n-2$ layer of the G_D via the present switch. Then, it will be passed along from the $n-2$ layer of the G_D to the destination switch, and ultimately to the destination node.

3.3. Packet Latency

We performed a small-scale simulation of the routing algorithms to evaluate the average packet latency of KANTC and MiKANTC. The simulation was performed with a k-value of 3 and an n-value of 4 and was evaluated in a uniform pattern per clock cycle. In the uniform traffic scenario, the destination addresses of packets were randomly assigned. The simulation only proceeded if there were available buffers in the switch for the given packet, otherwise, the packet would be delayed for a single clock cycle. The simulation assumed normal buffer size (NB) for all switches, as well as a double buffer size (DB) for intermediate switches. The traffic load, represented by λ, was varied in increments of 0.05 within the range of 0.05 and 1.00. In each clock cycle, $\lambda \times N$ computational nodes will communicate with uniform specified destinations via network datas, with N representing the quantity of nodes present within the system. The simulation concluded once each destination node received a mean of 200 packets.

4. Link Fault Tolerance

Link Fault Tolerance is an algorithm that ensures that in case of link fault in interconnection network topologies, the routing algorithm still ensures that the packets can reach the destination node. Many link fault tolerant routing algorithms have been developed to improve the reliability of the network, such as link backup, multipath routing. The method of link backup is to pre-configure multiple paths between nodes in case of emergencies such as link fault or congestion. When a link fault happens, the backup path will be used to replace the fault path to ensure timely arrival of data packets.

For larger-scale interconnection networks such as fat-tree, using link backups can significantly increase the hardware cost. Therefore, we focus on multipath algorithms for two topologies. We divided the link fault tolerant routing algorithms into two parts: hypercube and k-ary n-tree (Mirrored k-ary n-tree). The Hypercube architecture provides a high degree of interconnectivity between its nodes, so we can simply use the rerouting algorithm to re-search other paths from the present switch to the target switch when a link failure occurs. For the link fault tolerance algorithms of KANT and MiKANT, Wang and Li [11] proposed four algorithms for different cases. In order to achieve the better efficiency, we combine these different algorithms together to be able to switch different paths quickly for different link fault situations, which can ensure uninterrupted data transmission in the actual routing and thus achieve the purpose of traffic protection.

Algorithm 1: MiKANTC Link Fault Tolerance

Input: $packet = \langle D, data \rangle$,$linklist = \langle L_m, ..., L_0 \rangle$
if *packet in defferent group or cube* **then**
  send network data to SIS via $Link_P$;
  if $Link_P$ *is faulty* **then**
    use multipath of hypercube via $Link_T$;
    if $Link_T$ *is faulty* **then**
      failed routing;
    end
  else
    send network data to D^+_{LP} via $Link_P$;
    if $Link_P$ *is faulty* **then**
      use Go-Neighbor to P_N via $Link_T$;
      if $Link_T$ *is faulty* **then**
        use X-Turns to $N(D^+_{LP})$ via $Link_T$;
        if $Link_T$ *is faulty* **then**
          failed routing;
        end
      end
    else
      send network data to D^-_{LP} via $Link_P$;
      if $Link_P$ *is faulty* **then**
        use Go-Down to P_D via $Link_T$;
      end
    end
  end
else
  send network data to D via $Link_P$;
  if $Link_P$ *is faulty* **then**
    use multipath of hypercube via $Link_T$;
  else
    failed routing;
  end
end

Algorithm 1 gives the routing algorithm of MiKANTC link fault tolerance, when a packet arrives at a switch, the algorithm will first route according to the normal algorithm given before, and then route according to different situations when a link fault is detected. Where D is the destination switch, P is the present switch, SIS is the intermediate switch to the destination switch,the link will be routed to the intermediate switch through the shortest path if no link fault occurs, and will be routed to other paths and then to SIS when a fault occurs. D^+_{LP} is the

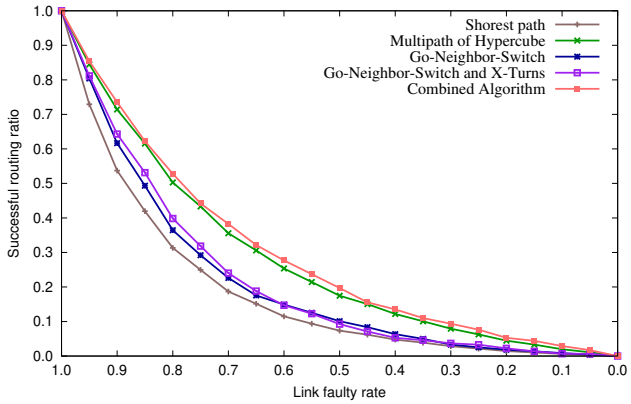

Figure 4: Successful routing ratio on MiKANTC$(3, 3)$ link fault tolerance

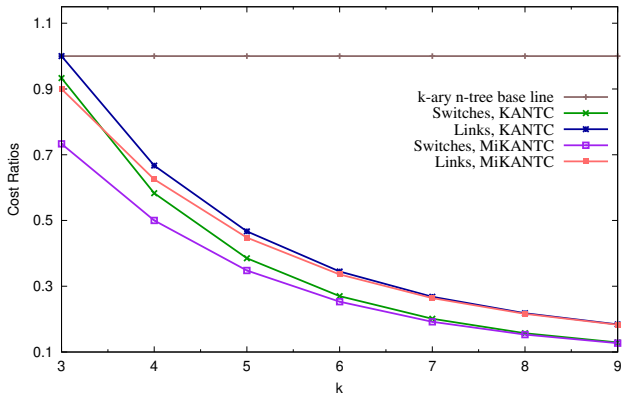

Figure 5: Cost ratios of switches and links to KANT(k, n)

label assigned to the port of switch P that connects to switches with the level equal to $L_P + 1$ (increased level). Similarly, D^{-}_{LP} connects to $L_P - 1$ (decreased level) level switches. P_N means the neighbor switch of P, P_D means the down-level switch of P, $N(D^{+}_{LP})$ means the another array of switches on the level $L_P + 1$ of the same group, reaching the destination switch by folding back(X). If a fault occurs, the Go-Neighbour Switch, X-Turns Routing Algorithm will be performed during the rise phase and the Go-Down-Level Algorithm will be performed during the fall phase, depending on the link fault.

Figure 4 shows the algorithm's routing success rate for link failures between 0% and 100%. We have tested its performance in a small interconnected network, for each algorithm we tested thousands of times. Combined algorithms, which is Algorithm 1 mentioned earlier in this paper, shows better performance.

5. Performance Evaluation

In this section, numerical simulations are performed to assess the hardware reduction, effectiveness, and path diversity of KANTC and MiKANTC.

5.1. Cost Ratio

We evaluate the cost of KANTC and MiKANTC relative to KANT and MiKANT, respectively. The cost of the topologies is characterized by the cost ratio of switch and

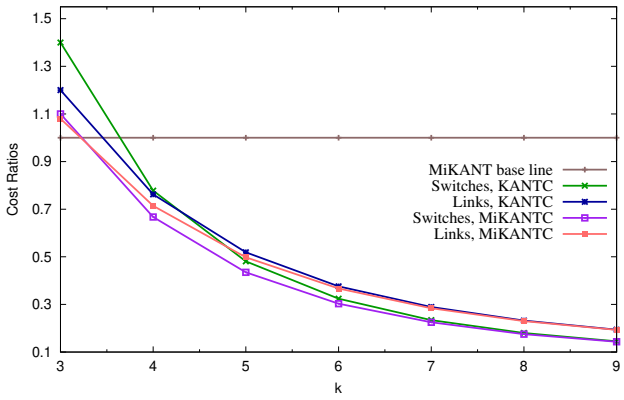

Figure 6: Cost ratios of switches and links to MiKANT(k, n)

link. Fig. 5 and fig. 6, show the cost ratio of switch and link, respectively, when n = k. The results demonstrate that, for the cases where $n = k \geq 4$, compared to KANT and MiKANT, both KANTC and MiKANTC provide a reduced cost ratio for hardware cost. When we set all parameters to 8, KANTC has a 84.27% reduction in switches and a 78.23% reduction in links compared to the KANT, and a 82.03% reduction in switches and a 76.77% reduction in links compared to MiKANT(k, n). Similarly, MiKANTC has a 84.68% reduction in switches and a 78.% reduction in links compared to the KANT, and with comparison to the MiKANT, the number of switches was reduced by 82.49% and the number of links by 76.99%.

5.2. Relative Cost Performance

The relative cost performance (RCP) of KANTC and MiKANTC has been defined to achieve an optimal balance between cost and performance, as follows:

$$RCP_1 = 2k \times (2n+k)(\log_2 N_1/p + p) \times (\log_2 N_1/p + 2)$$

$$RCP_2 = 2k \times (2n+k)(\log_2 N_2/p + p) \times (\log_2 N_2/p + 2)$$

The RCP of KANTC (k, n) and MiKANTC (k, n) are defined as RCP_1 and RCP_2, respectively. The RCP is influenced by factors such as the radix, diameter, size, and the number of ports in the router. The results are shown in Figures 7 and fig. 8, respectively, for the case of $2 \leq n \leq 7$ and $p = 1$. The results of the simulation demonstrated that lower values in the performance curves correspond to more efficient utilization of hardware resources. A value less than 1 suggests that the system outperforms the hypercube in terms of performance. For a system of 5412096 nodes with MiKANTC$(6, 7)$, for example, the RCP is 0.421. The results allow us to choose appropriate values of k and n for a given system size so that the RCP is minimised.

5.3. Path Diversities

Path diversity is a critical aspect of network topology that has a significant impact on the fault tolerance of the system. In this study, we define path diversity (PD) as the ratio of the mean of the shortest paths $(\widetilde{P})$ to total number of nodes (N) in the topology, calculated as

$$PD = \frac{\widetilde{P}}{N}$$

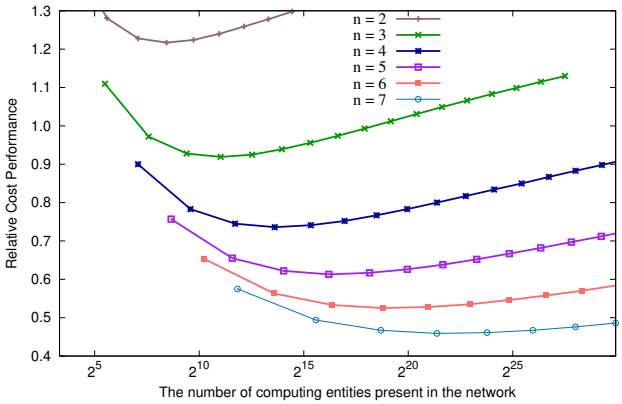

Figure 7: KANTC RCP performance

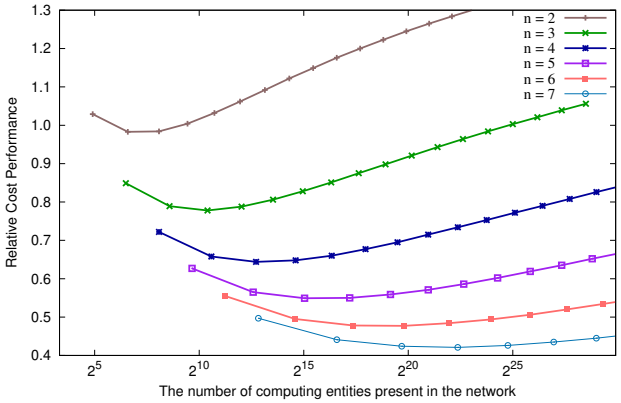

Figure 8: MiKANTC RCP performance

To assess the diversity of paths across various network configurations, a comparison of the k-cube, KANT, MiKANT, KANTC, and MiKANTC is depicted in Fig. 9, with n fixed at 8. The results demonstrate that the link diversity of KANTC and MiKANTC is superior to that of other topologies, as evidenced by the increased number of paths. This suggests that these two proposed networks exhibit improved routing performance and a higher level of fault tolerance compared to other existing network topologies.

6. Conclusions

Our research indicates that the KANTC and MiKANTC networks achieve a high performance-to-cost ratio and offer a higher level of path diversity compared to traditional fat-tree networks. Our future work will focus on developing algorithms with even better performance, which will be evaluated through simulations in a more realistic, large-scale network environment.

References

[1] W. Zheng, "Research trend of large-scale supercomputers and applications from the top500 and gordon bell prize," *Science China Information Sciences*, vol. 63, no. 7, pp. 1–14, 2020.

[2] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

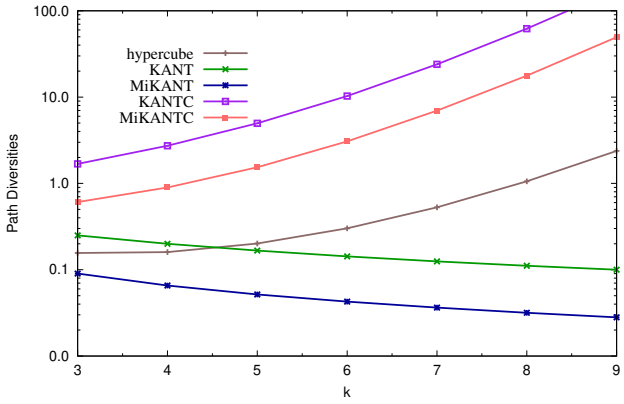

Figure 9: Path diversities

[3] J. Wells, B. Bland, J. Nichols, J. Hack, F. Foertter, G. Hagen, T. Maier, M. Ashfaq, B. Messer, and S. Parete-Koon, "Announcing supercomputer summit," Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), Tech. Rep., 2016.

[4] C. B. Stunkel, R. L. Graham, G. Shainer, M. Kagan, S. Sharkawi, B. Rosenburg, and G. Chochia, "The high-speed networks of the summit and sierra supercomputers," *IBM Journal of Research and Development*, vol. 64, no. 3/4, pp. 3–1, 2020.

[5] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proceedings 11th international parallel processing symposium*. IEEE, 1997, pp. 87–93.

[6] C. G. Requena, F. G. Villamón, M. E. G. Requena, P. J. L. Rodríguez, and J. D. Marín, "Ruft: Simplifying the fat-tree topology," in *2008 14th IEEE International Conference on Parallel and Distributed Systems*. IEEE, 2008, pp. 153–160.

[7] D. F. B. Garzón, C. G. Requena, M. E. Gómez, P. López, and J. Duato, "A family of fault-tolerant efficient indirect topologies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 927–940, 2015.

[8] D. Ludovici, F. Gilabert, C. Requena, M. Gmez, P. Lpez, G. Gaydadjiev, and J. Duato, "Butterfly vs. unidirectional fat-trees for networks-on-chip: Not a mere permutation of outputs," in *Proc. 3rd Workshop Interconnection Netw. Archit. On-Chip Multi-Chip*. Citeseer, 2009.

[9] Z. Wang, J. Xu, X. Wu, Y. Ye, W. Zhang, M. Nikdast, X. Wang, and Z. Wang, "Floorplan optimization of fat-tree-based networks-on-chip for chip multiprocessors," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1446–1459, 2012.

[10] Y. Li and W. Chu, "Mikant: A mirrored k-ary n-tree for reducing hardware cost and packet latency of fat-tree and clos networks," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2018, pp. 1643–1650.

[11] Y. Wang and Y. Li, "Link fault tolerant routing algorithms in mirrored k-ary n-tree interconnection networks," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2020, pp. 237–241.

[12] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Transactions on computers*, vol. 37, no. 7, pp. 867–872, 1988.

[13] D. Arai and Y. Li, "Generalized-star cube: A new class of interconnection topology for massively parallel systems," in *2015 Third International Symposium on Computing and Networking (CANDAR)*. IEEE, 2015, pp. 68–74.

[14] Y. Wang and Y. Li, "Hybrid interconnection networks for reducing hardware cost and improving path diversity based on fat-trees and hypercubes," in *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2021, pp. 254–260.